django-axes documentation

Release 5.26.1.dev2+g5dda717

Jazzband

Oct 13, 2021

Contents

CHAPTER 1

Contents

1.1 Requirements

Axes requires a supported Django version and runs on Python versions 3.6 and above.

Refer to the project source code repository in GitHub and see the Tox configuration and Python package definition to check if your Django and Python version are supported.

The GitHub Actions builds test Axes compatibility with the Django master branch for future compatibility as well.

1.2 Installation

Axes is easy to install from the PyPI package:

```
$ pip install django-axes
```

After installing the package, the project settings need to be configured.

1. Add `axes` to your `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

(continues on next page)

(continued from previous page)

```
    # Axes app can be in any position in the INSTALLED_APPS list.
    'axes',
]
```

2. Add `axes.backends.AxesBackend` to the top of `AUTHENTICATION_BACKENDS`:

```
AUTHENTICATION_BACKENDS = [
    # AxesBackend should be the first backend in the AUTHENTICATION_
↪BACKENDS list.
    'axes.backends.AxesBackend',

    # Django ModelBackend is the default authentication backend.
    'django.contrib.auth.backends.ModelBackend',
]
```

3. Add `axes.middleware.AxesMiddleware` to your list of `MIDDLEWARE`:

```
MIDDLEWARE = [
    # The following is the list of default middleware in new Django␣
↪projects.
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    # AxesMiddleware should be the last middleware in the␣
↪MIDDLEWARE list.
    # It only formats user lockout messages and renders Axes␣
↪lockout responses
    # on failed user authentication attempts from login views.
    # If you do not want Axes to override the authentication␣
↪response
    # you can skip installing the middleware and use your own views.
    'axes.middleware.AxesMiddleware',
]
```

4. Run `python manage.py check` to check the configuration.

5. Run `python manage.py migrate` to sync the database.

Axes is now functional with the default settings and is saving user attempts into your database and locking users out if they exceed the maximum attempts.

You should use the `python manage.py check` command to verify the correct configuration in development, staging, and production environments. It is probably best to use this step as part of your regular CI workflows to verify that your project is not misconfigured.

Axes uses checks to verify your Django settings configuration for security and functionality.

Many people have different configurations for their development and production environments, and running the application with misconfigured settings can prevent security features from working.

1.2.1 Disabling Axes system checks

If you are implementing custom authentication, request middleware, or signal handlers the Axes checks system might generate false positives in the Django checks framework.

You can silence the unnecessary warnings by using the following Django settings:

```
SILENCED_SYSTEM_CHECKS = ['axes.W003']
```

Axes has the following warnings codes built in:

- `axes.W001` for invalid `CACHES` configuration.
- `axes.W002` for invalid `MIDDLEWARE` configuration.
- `axes.W003` for invalid `AUTHENTICATION_BACKENDS` configuration.
- `axes.W004` for deprecated use of `AXES_*` setting flags.

Note: Only disable the Axes system checks and warnings if you know what you are doing. The default checks are implemented to verify and improve your project's security and should only produce necessary warnings due to misconfigured settings.

1.2.2 Disabling Axes components in tests

If you get errors when running tests, try setting the `AXES_ENABLED` flag to `False` in your test settings:

```
AXES_ENABLED = False
```

This disables the Axes middleware, authentication backend and signal receivers, which might fix errors with incompatible test configurations.

1.2.3 Disabling atomic requests

Django offers atomic database transactions that are tied to HTTP requests and toggled on and off with the `ATOMIC_REQUESTS` configuration.

When `ATOMIC_REQUESTS` is set to `True` Django will always either perform all database read and write operations in one successful atomic transaction or in a case of failure roll them back, leaving no trace of the failed request in the database.

However, sometimes Axes or another plugin can misbehave or not act correctly with other code, preventing the login mechanisms from working due to e.g. exception being thrown in some part

of the code, preventing access attempts being logged to database with Axes or causing similar problems.

If new attempts or log objects are not being correctly written to the Axes tables, it is possible to configure Django `ATOMIC_REQUESTS` setting to to `False`:

```
ATOMIC_REQUESTS = False
```

Please note that atomic requests are usually desirable when writing e.g. RESTful APIs, but sometimes it can be problematic and warrant a disable.

Before disabling atomic requests or configuring them please read the relevant Django documentation and make sure you know what you are configuring rather than just toggling the flag on and off for testing.

Also note that the cache backend can provide correct functionality with Memcached or Redis caches even with exceptions being thrown in the stack.

1.3 Usage

Once Axes is installed and configured, you can login and logout of your application via the `django.contrib.auth` views. The attempts will be logged and visible in the Access Attempts section in admin.

Axes monitors the views by using the Django login and logout signals and locks out user attempts with a custom authentication backend that checks if requests are allowed to authenticate per the configured rules.

By default, Axes will lock out repeated access attempts from the same IP address by monitoring login failures and storing them into the default database.

1.3.1 Authenticating users

Axes needs a `request` attribute to be supplied to the stock Django `authenticate` method in the `django.contrib.auth` module in order to function correctly.

If you wish to manually supply the argument to the calls to `authenticate`, you can use the following snippet in your custom login views, tests, or other code:

```
def custom_login_view(request)
    username = ...
    password = ...

    user = authenticate(
        request=request,  # this is the important custom argument
        username=username,
        password=password,
    )
```

(continues on next page)

(continued from previous page)

```
    if user is not None:
        login(request, user)
```

If your test setup has problems with the `request` argument, you can either supply the argument manually with a blank *HttpRequest()*' object, disable Axes in the test setup by excluding `axes` from `INSTALLED_APPS`, or leave out `axes.backends.AxesBackend` from your `AUTHENTICATION_BACKENDS`.

If you are using a 3rd party library that does not supply the `request` attribute when calling `authenticate` you can implement a customized backend that inherits from `axes.backends.AxesBackend` or other backend and overrides the `authenticate` method.

1.3.2 Resetting attempts and lockouts

When Axes locks an IP address, it is not allowed to login again. You can allow IPs to attempt again by resetting (deleting) the relevant AccessAttempt records in the admin UI, CLI, or your own code.

You can also configure automatic cool down periods, IP whitelists, and custom code and handler functions for resetting attempts. Please check out the configuration and customization documentation for further information.

Note: Please note that the functionality describe here concerns the default database handler. If you have changed the default handler to another class such as the cache handler you have to implement custom reset commands.

Resetting attempts from the Django admin UI

Records can be easily deleted by using the Django admin application.

Go to the admin UI and check the `Access Attempt` view. Select the attempts you wish the allow again and simply remove them. The blocked user will be allowed to log in again in accordance to the rules.

Resetting attempts from command line

Axes offers a command line interface with `axes_reset`, `axes_reset_ip`, and `axes_reset_username` management commands with the Django `manage.py` or `django-admin` command helpers:

- `python manage.py axes_reset` will reset all lockouts and access records.
- `python manage.py axes_reset_ip [ip ...]` will clear lockouts and records for the given IP addresses.

- `python manage.py axes_reset_username [username ...]` will clear lockouts and records for the given usernames.
- `python manage.py axes_reset_logs (age)` will reset (i.e. delete) AccessLog records that are older than the given age where the default is 30 days.

Resetting attempts programmatically by APIs

In your code, you can use the `axes.utils.reset` function.

- `reset()` will reset all lockouts and access records.
- `reset(ip=ip)` will clear lockouts and records for the given IP address.
- `reset(username=username)` will clear lockouts and records for the given username.

Note: Please note that if you give both `username` and `ip` arguments to `reset` that attempts that have both the set IP and username are reset. The effective behaviour of `reset` is to `and` the terms instead of `or` ing them.

1.4 Configuration

Minimal Axes configuration is done with just `settings.py` updates.

More advanced configuration and integrations might require updates on source code level depending on your project implementation.

1.4.1 Configuring project settings

The following `settings.py` options are available for customizing Axes behaviour.

- `AXES_ENABLED`: Enable or disable Axes plugin functionality, for example in test runner setup. Default: `True`
- `AXES_FAILURE_LIMIT`: The integer number of login attempts allowed before a record is created for the failed logins. This can also be a callable or a dotted path to callable that returns an integer and all of the following are valid: `AXES_FAILURE_LIMIT = 42`, `AXES_FAILURE_LIMIT = lambda *args: 42`, and `AXES_FAILURE_LIMIT = 'project.app.get_login_failure_limit'`. Default: `3`
- `AXES_LOCK_OUT_AT_FAILURE`: After the number of allowed login attempts are exceeded, should we lock out this IP (and optional user agent)? Default: `True`
- `AXES_COOLOFF_TIME`: If set, defines a period of inactivity after which old failed login attempts will be cleared. Can be set to a Python timedelta object, an integer, a callable,

or a string path to a callable which takes no arguments. If an integer, will be interpreted as a number of hours. Default: `None`

- `AXES_ONLY_ADMIN_SITE`: If `True`, lock is only enabled for admin site. Admin site is determined by checking request path against the path of `"admin:index"` view. If admin urls are not registered in current urlconf, all requests will not be locked. Default: `False`
- `AXES_ONLY_USER_FAILURES` : If `True`, only lock based on username, and never lock based on IP if attempts exceed the limit. Otherwise utilize the existing IP and user locking logic. Default: `False`
- `AXES_ENABLE_ADMIN`: If `True`, admin views for access attempts and logins are shown in Django admin interface. Default: `True`
- `AXES_LOCK_OUT_BY_COMBINATION_USER_AND_IP`: If `True`, prevent login from IP under a particular username if the attempt limit has been exceeded, otherwise lock out based on IP. Default: `False`
- `AXES_LOCK_OUT_BY_USER_OR_IP`: If `True`, prevent login from if the attempt limit has been exceeded for IP or username. Default: `False`
- `AXES_USE_USER_AGENT`: If `True`, lock out and log based on the IP address and the user agent. This means requests from different user agents but from the same IP are treated differently. This settings has no effect if the `AXES_ONLY_USER_FAILURES` setting is active. Default: `False`
- `AXES_HANDLER`: The path to the handler class to use. If set, overrides the default signal handler backend. Default: `'axes.handlers.database.AxesDatabaseHandler'`
- `AXES_CACHE`: The name of the cache for Axes to use. Default: `'default'`
- `AXES_LOCKOUT_TEMPLATE`: If set, specifies a template to render when a user is locked out. Template receives `cooloff_timedelta`, `cooloff_time`, `username` and `failure_limit` as context variables. Default: `None`
- `AXES_LOCKOUT_URL`: If set, specifies a URL to redirect to on lockout. If both `AXES_LOCKOUT_TEMPLATE` and `AXES_LOCKOUT_URL` are set, the template will be used. Default: `None`
- `AXES_VERBOSE`: If `True`, you'll see slightly more logging for Axes. Default: `True`
- `AXES_USERNAME_FORM_FIELD`: the name of the form field that contains your users usernames. Default: `username`
- `AXES_USERNAME_CALLABLE`: A callable or a string path to callable that takes two arguments for user lookups: `def get_username(request: HttpRequest, credentials: dict) -> str: ...` This can be any callable such as `AXES_USERNAME_CALLABLE = lambda request, credentials: 'username'` or a full Python module path to callable such as `AXES_USERNAME_CALLABLE = 'example.get_username`. The `request` is a HttpRequest like object and the `credentials` is a dictionary like object. `credentials` are the ones that were passed to Django `authenticate()` in the lo-

gin flow. If no function is supplied, Axes fetches the username from the `credentials` or `request.POST` dictionaries based on `AXES_USERNAME_FORM_FIELD`.

- `AXES_WHITELIST_CALLABLE`: A callable or a string path to callable that takes two arguments for whitelisting determination and returns True, if user should be whitelisted: `def is_whilisted(request: HttpRequest, credentials: dict) -> bool: ...`. This can be any callable similarly to `AXES_USERNAME_CALLABLE`. Default: `None`
- `AXES_LOCKOUT_CALLABLE`: A callable or a string path to callable that takes two arguments returns a response. For example: `def generate_lockout_response(request: HttpRequest, credentials: dict) -> HttpResponse: ...`. This can be any callable similarly to `AXES_USERNAME_CALLABLE`. If not callable is defined, then the default implementation in `axes.helpers.get_lockout_response` is used for determining the correct lockout response that is sent to the requesting client. Default: `None`
- `AXES_PASSWORD_FORM_FIELD`: the name of the form or credentials field that contains your users password. Default: `password`
- `AXES_SENSITIVE_PARAMETERS`: Configures POST and GET parameter values (in addition to the value of `AXES_PASSWORD_FORM_FIELD`) to mask in login attempt logging. Default: `[]`
- `AXES_NEVER_LOCKOUT_GET`: If `True`, Axes will never lock out HTTP GET requests. Default: `False`
- `AXES_NEVER_LOCKOUT_WHITELIST`: If `True`, users can always login from whitelisted IP addresses. Default: `False`
- `AXES_IP_BLACKLIST`: An iterable of IPs to be blacklisted. Takes precedence over whitelists. For example: `AXES_IP_BLACKLIST = ['0.0.0.0']`. Default: `None`
- `AXES_IP_WHITELIST`: An iterable of IPs to be whitelisted. For example: `AXES_IP_WHITELIST = ['0.0.0.0']`. Default: `None`
- `AXES_DISABLE_ACCESS_LOG`: If `True`, disable writing login and logout access logs to database, so the admin interface will not have user login trail for successful user authentication. Default: `False`
- `AXES_RESET_ON_SUCCESS`: If `True`, a successful login will reset the number of failed logins. Default: `False`
- `AXES_ALLOWED_CORS_ORIGINS`: Configures lockout response CORS headers for XHR requests. Default: `*`
- `AXES_HTTP_RESPONSE_CODE`: Sets the http response code returned when `AXES_FAILURE_LIMIT` is reached. For example: `AXES_HTTP_RESPONSE_CODE = 429` Default: `403`

The configuration option precedences for the access attempt monitoring are:

1. Default: only use IP address.

2. `AXES_ONLY_USER_FAILURES`: only user username (`AXES_USE_USER_AGENT` has no effect).
3. `AXES_LOCK_OUT_BY_COMBINATION_USER_AND_IP`: use username and IP address.

The `AXES_USE_USER_AGENT` setting can be used with username and IP address or just IP address monitoring, but does nothing when the `AXES_ONLY_USER_FAILURES` setting is set.

1.4.2 Configuring reverse proxies

Axes makes use of `django-ipware` package to detect the IP address of the client and uses some conservative configuration parameters by default for security.

If you are using reverse proxies, you will need to configure one or more of the following settings to suit your set up to correctly resolve client IP addresses:

- `AXES_PROXY_COUNT`: The number of reverse proxies in front of Django as an integer. Default: `None`
- `AXES_META_PRECEDENCE_ORDER`: The names of `request.META` attributes as a tuple of strings to check to get the client IP address. Check the Django documentation for header naming conventions. Default: `IPWARE_META_PRECEDENCE_ORDER` setting if set, else `('REMOTE_ADDR', )`

Note: For reverse proxies or e.g. Heroku, you might also want to fetch IP addresses from a HTTP header such as `X-Forwarded-For`. To configure this, you can fetch IPs through the `HTTP_X_FORWARDED_FOR` key from the `request.META` property which contains all the HTTP headers in Django:

```
# refer to the Django request and response objects documentation
AXES_META_PRECEDENCE_ORDER = [
    'HTTP_X_FORWARDED_FOR',
    'REMOTE_ADDR',
]
```

Please note that proxies have different behaviours with the HTTP headers. Make sure that your proxy either strips the incoming value or otherwise makes sure of the validity of the header that is used because **any header values used in application configuration must be secure and trusted**. Otherwise the client can spoof IP addresses by just setting the header in their request and circumvent the IP address monitoring. Normal proxy server behaviours include overriding and appending the header value depending on the platform. Different platforms and gateway services utilize different headers, please refer to your deployment target documentation for up-to-date information on correct configuration.

1.4.3 Configuring handlers

Axes uses handlers for processing signals and events from Django authentication and login attempts.

The following handlers are implemented by Axes and can be configured with the `AXES_HANDLER` setting in project configuration:

- `axes.handlers.database.AxesDatabaseHandler` logs attempts to database and creates AccessAttempt and AccessLog records that persist until removed from the database manually or automatically after their cool offs expire (checked on each login event).
- `axes.handlers.cache.AxesCacheHandler` only uses the cache for monitoring attempts and does not persist data other than in the cache backend; this data can be purged automatically depending on your cache configuration, so the cache handler is by design less secure than the database backend but offers higher throughput and can perform better with less bottlenecks. The cache backend should ideally be used with a central cache system such as a Memcached cache and should not rely on individual server state such as the local memory or file based cache does.
- `axes.handlers.dummy.AxesDummyHandler` does nothing with attempts and can be used to disable Axes handlers if the user does not wish Axes to execute any logic on login signals. Please note that this effectively disables any Axes security features, and is meant to be used on e.g. local development setups and testing deployments where login monitoring is not wanted.

To switch to cache based attempt tracking you can do the following:

```
AXES_HANDLER = 'axes.handlers.cache.AxesCacheHandler'
```

See the cache configuration section for suitable cache backends.

1.4.4 Configuring caches

If you are running Axes with the cache based handler on a deployment with a local Django cache, the Axes lockout and reset functionality might not work predictably if the cache in use is not the same for all the Django processes.

Axes needs to cache access attempts application-wide, and e.g. the in-memory cache only caches access attempts per Django process, so for example resets made in the command line might not remove lock-outs that are in a separate process's in-memory cache such as the web server serving your login or admin page.

To circumvent this problem, please use somethings else than `django.core.cache.backends.dummy.DummyCache`, `django.core.cache.backends.locmem.LocMemCache`, or `django.core.cache.backends.filebased.FileBasedCache` as your cache backend in Django cache `BACKEND` setting.

If changing the `'default'` cache is not an option, you can add a cache specifically for use with Axes. This is a two step process. First you need to add an extra cache to `CACHES` with a

name of your choice:

```
CACHES = {
    'axes': {
        'BACKEND': 'django.core.cache.backends.memcached.
↪MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

The next step is to tell Axes to use this cache through adding `AXES_CACHE` to your `settings.py` file:

```
AXES_CACHE = 'axes'
```

There are no known problems in e.g. `MemcachedCache` or Redis based caches.

1.4.5 Configuring authentication backends

Axes requires authentication backends to pass request objects with the authentication requests for performing monitoring.

If you get `AxesBackendRequestParameterRequired` exceptions, make sure any libraries and middleware you use pass the request object.

Please check the integration documentation for further information.

1.4.6 Configuring 3rd party apps

Refer to the integration documentation for Axes configuration with third party applications and plugins such as

- Django REST Framework
- Django Allauth
- Django Simple Captcha

1.5 Customization

Axes has multiple options for customization including customizing the attempt tracking and lockout handling logic and lockout response formatting.

There are public APIs and the whole Axes tracking system is pluggable. You can swap the authentication backend, attempt tracker, failure handlers, database or cache backends and error formatters as you see fit.

Check the API reference section for further inspiration on implementing custom authentication backends, middleware, and handlers.

Axes uses the stock Django signals for login monitoring and can be customized and extended by using them correctly.

Axes listens to the following signals from `django.contrib.auth.signals` to log access attempts:

- `user_logged_in`
- `user_logged_out`
- `user_login_failed`

You can also use Axes with your own auth module, but you'll need to ensure that it sends the correct signals in order for Axes to log the access attempts.

1.5.1 Customizing authentication views

Here is a more detailed example of sending the necessary signals using and a custom auth backend at an endpoint that expects JSON requests. The custom authentication can be swapped out with `authenticate` and `login` from `django.contrib.auth`, but beware that those methods take care of sending the nessary signals for you, and there is no need to duplicate them as per the example.

`example/forms.py`:

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField(max_length=128, required=True)
    password = forms.CharField(max_length=128, required=True)
```

`example/views.py`:

```
from django.contrib.auth import signals
from django.http import JsonResponse, HttpResponse
from django.utils.decorators import method_decorator
from django.views import View
from django.views.decorators.csrf import csrf_exempt

from axes.decorators import axes_dispatch

from example.forms import LoginForm
from example.authentication import authenticate, login


@method_decorator(axes_dispatch, name='dispatch')
@method_decorator(csrf_exempt, name='dispatch')
class Login(View):
    """
    Custom login view that takes JSON credentials
    """
```

(continues on next page)

(continued from previous page)

```
    http_method_names = ['post']

    def post(self, request):
        form = LoginForm(request.POST)

        if not form.is_valid():
            # inform django-axes of failed login
            signals.user_login_failed.send(
                sender=User,
                request=request,
                credentials={
                    'username': form.cleaned_data.get('username'),
                },
            )
            return HttpResponse(status=400)

        user = authenticate(
            request=request,
            username=form.cleaned_data.get('username'),
            password=form.cleaned_data.get('password'),
        )

        if user is not None:
            login(request, user)

            signals.user_logged_in.send(
                sender=User,
                request=request,
                user=user,
            )

            return JsonResponse({
                'message':'success'
            }, status=200)

        # inform django-axes of failed login
        signals.user_login_failed.send(
            sender=User,
            request=request,
            credentials={
                'username': form.cleaned_data.get('username'),
            },
        )

        return HttpResponse(status=403)
```

urls.py:

```
from django.urls import path
from example.views import Login

urlpatterns = [
    path('login/', Login.as_view(), name='login'),
]
```

1.5.2 Customizing username lookups

In special cases, you may have the need to modify the username that is submitted before attempting to authenticate. For example, adding namespacing or removing client-set prefixes. In these cases, `axes` needs to know how to make these changes so that it can correctly identify the user without any form cleaning or validation. This is where the `AXES_USERNAME_CALLABLE` setting comes in. You can define how to make these modifications in a callable that takes a request object and a credentials dictionary, and provide that callable to `axes` via this setting.

For example, a function like this could take a post body with something like `username='prefixed-username'` and `namespace=my_namespace` and turn it into `my_namespace-username`:

`example/utils.py`:

```
def get_username(request, credentials):
    username = credentials.get('username')
    namespace = credentials.get('namespace')
    return namespace + '-' + username
```

`settings.py`:

```
AXES_USERNAME_CALLABLE = 'example.utils.get_username'
```

Note: You still have to make these modifications yourself before calling authenticate. If you want to re-use the same function for consistency, that's fine, but Axes does not inject these changes into the authentication flow for you.

1.5.3 Customizing lockout responses

Axes can be configured with `AXES_LOCKOUT_CALLABLE` to return a custom lockout response when using the plugin with e.g. DRF (Django REST Framework) or other third party libraries which require specialized formats such as JSON or XML response formats or customized response status codes.

An example of usage could be e.g. a custom view for processing lockouts.

`example/views.py`:

```
from django.http import JsonResponse

def lockout(request, credentials, *args, **kwargs):
    return JsonResponse({"status": "Locked out due to too many␣
↪login failures"}, status=403)
```

settings.py:

```
AXES_LOCKOUT_CALLABLE = "example.views.lockout"
```

1.6 Integration

Axes is intended to be pluggable and usable with custom authentication solutions. This document describes the integration with some popular 3rd party packages such as Django Allauth, Django REST Framework, and other tools.

In the following table **Compatible** means that a component should be fully compatible out-of-the-box, **Functional** means that a component should be functional after configuration, and **Incompatible** means that a component has been reported as non-functional with Axes.

Project	Version	Compatible	Functional	Incompatible
Django REST Framework			✓	
Django Allauth			✓	
Django Simple Captcha			✓	
Django OAuth Toolkit			✓	
Django Reversion			✓	

Please note that project compatibility depends on multiple different factors such as Django version, Axes version, and 3rd party package versions and their unique combinations per project.

Note: This documentation is mostly provided by Axes users. If you have your own compatibility tweaks and customizations that enable you to use Axes with other tools or have better implementations than the solutions provided here, please do feel free to open an issue or a pull request in GitHub!

1.6.1 Integration with Django Allauth

Axes relies on having login information stored under `AXES_USERNAME_FORM_FIELD` key both in `request.POST` and in `credentials` dict passed to `user_login_failed` signal.

This is not the case with Allauth. Allauth always uses the `login` key in post POST data but it becomes `username` key in `credentials` dict in signal handler.

To overcome this you need to use custom login form that duplicates the value of `username` key under a `login` key in that dict and set `AXES_USERNAME_FORM_FIELD = 'login'`.

You also need to decorate `dispatch()` and `form_invalid()` methods of the Allauth login view.

`settings.py`:

```
AXES_USERNAME_FORM_FIELD = 'login'
```

`example/forms.py`:

```
from allauth.account.forms import LoginForm

class AxesLoginForm(LoginForm):
    """
    Extended login form class that supplied the
    user credentials for Axes compatibility.
    """

    def user_credentials(self):
        credentials = super().user_credentials()
        credentials['login'] = credentials.get('email') or␣
↪credentials.get('username')
        return credentials
```

`example/urls.py`:

```
from django.utils.decorators import method_decorator

from allauth.account.views import LoginView

from axes.decorators import axes_dispatch
from axes.decorators import axes_form_invalid

from example.forms import AxesLoginForm

LoginView.dispatch = method_decorator(axes_dispatch)(LoginView.
↪dispatch)
LoginView.form_invalid = method_decorator(axes_form_
↪invalid)(LoginView.form_invalid)

urlpatterns = [
    # Override allauth default login view with a patched view
    path('accounts/login/', LoginView.as_view(form_
↪class=AxesLoginForm), name='account_login'),
    path('accounts/', include('allauth.urls')),
]
```

1.6.2 Integration with Django REST Framework

Django Axes requires REST Framework to be connected via lockout signals for correct functionality.

You can use the following snippet in your project signals such as `example/signals.py`:

```
from django.dispatch import receiver

from axes.signals import user_locked_out
from rest_framework.exceptions import PermissionDenied


@receiver(user_locked_out)
def raise_permission_denied(*args, **kwargs):
    raise PermissionDenied("Too many failed login attempts")
```

And then configure your application to load it in `examples/apps.py`:

```
from django import apps


class AppConfig(apps.AppConfig):
    name = "example"

    def ready(self):
        from example import signals  # noqa
```

Please check the Django signals documentation for more information:

https://docs.djangoproject.com/en/3.1/topics/signals/

When a user login fails a signal is emitted and PermissionDenied raises a HTTP 403 reply which interrupts the login process.

This functionality was handled in the middleware for a time, but that resulted in extra database requests being made for each and every web request, and was migrated to signals.

1.6.3 Integration with Django Simple Captcha

Axes supports Captcha with the Django Simple Captcha package in the following manner.

`settings.py`:

```
AXES_LOCKOUT_URL = '/locked'
```

`example/urls.py`:

```
url(r'^locked/$', locked_out, name='locked_out'),
```

`example/forms.py`:

```
class AxesCaptchaForm(forms.Form):
    captcha = CaptchaField()
```

example/views.py:

```
from axes.utils import reset_request
from django.http.response import HttpResponseRedirect
from django.shortcuts import render
from django.urls import reverse_lazy

from .forms import AxesCaptchaForm


def locked_out(request):
    if request.POST:
        form = AxesCaptchaForm(request.POST)
        if form.is_valid():
            reset_request(request)
            return HttpResponseRedirect(reverse_lazy('auth_login'))
    else:
        form = AxesCaptchaForm()

    return render(request, 'accounts/captcha.html', {'form': form})
```

example/templates/example/captcha.html:

```
<form action="" method="post">
    {% csrf_token %}

    {{ form.captcha.errors }}
    {{ form.captcha }}

    <div class="form-actions">
        <input type="submit" value="Submit" />
    </div>
</form>
```

1.6.4 Integration with Django OAuth Toolkit

Django OAuth toolkit is not designed to work with Axes, but some users have reported that they have configured validator classes to function correctly.

example/validators.py:

```
from django.contrib.auth import authenticate
from django.http import HttpRequest, QueryDict

from oauth2_provider.oauth2_validators import OAuth2Validator
```

(continues on next page)

(continued from previous page)

```
from axes.helpers import get_client_ip_address, get_client_user_
↪agent


class AxesOAuth2Validator(OAuth2Validator):
    def validate_user(self, username, password, client, request,␣
↪*args, **kwargs):
        """
        Check username and password correspond to a valid and␣
↪active User

        Set defaults for necessary request object attributes for␣
↪Axes compatibility.
        The ``request`` argument is not a Django ``HttpRequest``␣
↪object.
        """

        _request = request
        if request and not isinstance(request, HttpRequest):
            request = HttpRequest()

            request.uri = _request.uri
            request.method = request.http_method = _request.http_
↪method
            request.META = request.headers = _request.headers
            request._params = _request._params
            request.decoded_body = _request.decoded_body

            request.axes_ip_address = get_client_ip_address(request)
            request.axes_user_agent = get_client_user_agent(request)

            body = QueryDict(str(_request.body), mutable=True)
            if request.method == 'GET':
                request.GET = body
            elif request.method == 'POST':
                request.POST = body

        user = authenticate(request=request, username=username,␣
↪password=password)
        if user is not None and user.is_active:
            request = _request
            request.user = user
            return True

        return False
```

settings.py:

```
OAUTH2_PROVIDER = {
```

(continues on next page)

(continued from previous page)

```
    'OAUTH2_VALIDATOR_CLASS': 'example.validators.
↪AxesOAuth2Validator',
    'SCOPES': {'read': 'Read scope', 'write': 'Write scope'},
}
```

1.6.5 Integration with Django Reversion

Django Reversion is not designed to work with Axes, but some users have reported that they have configured a workaround with a monkeypatch function that functions correctly.

`example/monkeypatch.py`:

```
from django.urls import resolve

from reversion import views

def _request_creates_revision(request):
    view_name = resolve(request.path_info).url_name
    if view_name and view_name.endswith('login'):
        return False

    return request.method not in ["OPTIONS", "GET", "HEAD"]

views._request_creates_revision = _request_creates_revision
```

1.7 Architecture

Axes is based on the existing Django authentication backend architecture and framework for recognizing users and aims to be compatible with the stock design and implementation of Django while offering extensibility and configurability for using the Axes authentication monitoring and logging for users of the package as well as 3rd party package vendors such as Django REST Framework, Django Allauth, Python Social Auth and so forth.

The development of custom 3rd party package support are active goals, but you should check the up-to-date documentation and implementation of Axes for current compatibility before using Axes with custom solutions and make sure that authentication monitoring is working correctly.

This document describes the Django authentication flow and how Axes augments it to achieve authentication and login monitoring and lock users out on too many access attempts.

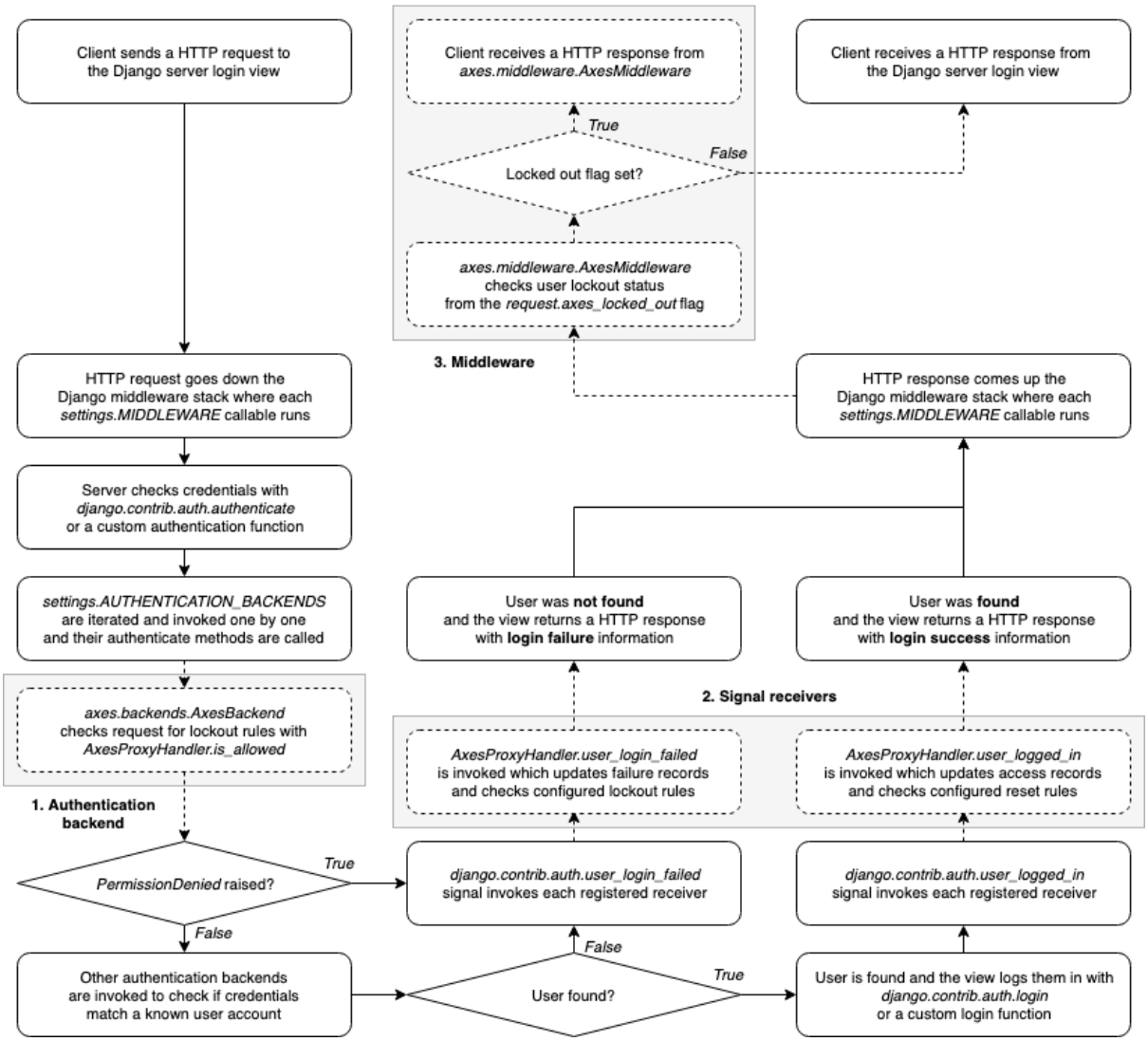
1.7.1 Django Axes authentication flow

Axes offers a few additions to the Django authentication flow that implement the login monitoring and lockouts through a swappable **handler** API and configuration flags that users and

package vendors can use to customize Axes or their own projects as they best see fit.

The following diagram visualizes the Django login flow and highlights the following extra steps that Axes adds to it with the **1. Authentication backend**, **2. Signal receivers**, and **3. Middleware**.

Django Axes augmented authentication flow with custom authentication backend, signal receivers, and middleware



When a user tries to log in in Django, the login is usually performed by running a number of authentication backends that check user login information by calling the `authenticate` function, which either returns a Django compatible `User` object or a `None`.

If an authentication backend does not approve a user login, it can raise a `PermissionDenied` exception, which immediately skips the rest of the authentication backends, triggers the `user_login_failed` signal, and then returns a `None` to the calling function, indicating that the login failed.

Axes implements authentication blocking with the custom `AxesBackend` authentication backend which checks every request coming through the Django authentication flow and verifies they are not blocked, and allows the requests to go through if the check passes.

If the authentication attempt matches a lockout rule, e.g. it is from a blacklisted IP

or exceeds the maximum configured authentication attempts, it is blocked by raising the `PermissionDenied` excepton in the backend.

Axes monitors logins with the `user_login_failed` signal receiver and records authentication failures from both the `AxesBackend` and other authentication backends and tracks the failed attempts by tracking the attempt IP address, username, user agent, or all of them.

If the lockout rules match, then Axes marks the request as locked by setting a special attribute into the request. The `AxesMiddleware` then processes the request, returning a lockout response to the user, if the flag has been set.

Axes assumes that the login views either call the `authenticate` method to log in users or otherwise take care of notifying Axes of authentication attempts and failures the same way Django does via authentication signals.

The login flows can be customized and the Axes authentication backend, middleware, and signal receivers can easily be swapped to alternative implementations.

1.8 API reference

Axes offers extensible APIs that you can customize to your liking. You can specialize the following base classes or alternatively use third party modules as long as they implement the following APIs.

class `axes.handlers.base.`**AbstractAxesHandler**
Contract that all handlers need to follow

get_failures(*request*, *credentials: dict = None*) → int
Checks the number of failures associated to the given request and credentials.

This is a virtual method that needs an implementation in the handler subclass if the `settings.AXES_LOCK_OUT_AT_FAILURE` flag is set to `True`.

user_logged_in(*sender*, *request*, *user*, ***kwargs*)
Handles the Django `django.contrib.auth.signals.user_logged_in` authentication signal.

user_logged_out(*sender*, *request*, *user*, ***kwargs*)
Handles the Django `django.contrib.auth.signals.user_logged_out` authentication signal.

user_login_failed(*sender*, *credentials: dict*, *request=None*, ***kwargs*)
Handles the Django `django.contrib.auth.signals.user_login_failed` authentication signal.

class `axes.handlers.base.`**AxesBaseHandler**
Handler API definition for implementations that are used by the `AxesProxyHandler`.

If you wish to specialize your own handler class, override the necessary methods and configure the class for use by setting `settings.AXES_HANDLER = 'module.path.to.YourClass'`. Make sure that new the handler is compliant with AbstractAxesHandler and make sure it extends from this mixin. Refer to *AxesHandler* for an example.

The default implementation that is actually used by Axes is `axes.handlers.database.AxesDatabaseHandler`.

Note: This is a virtual class and **can not be used without specialization**.

---

is_admin_site(*request*) → bool
Checks if the request is for admin site.

is_allowed(*request*, *credentials: dict = None*) → bool
Checks if the user is allowed to access or use given functionality such as a login view or authentication.

This method is abstract and other backends can specialize it as needed, but the default implementation checks if the user has attempted to authenticate into the site too many times through the Django authentication backends and returns `False` if user exceeds the configured Axes thresholds.

This checker can implement arbitrary checks such as IP whitelisting or blacklisting, request frequency checking, failed attempt monitoring or similar functions.

Please refer to the `axes.handlers.database.AxesDatabaseHandler` for the default implementation and inspiration on some common checks and access restrictions before writing your own implementation.

is_blacklisted(*request*, *credentials: dict = None*) → bool
Checks if the request or given credentials are blacklisted from access.

is_locked(*request*, *credentials: dict = None*) → bool
Checks if the request or given credentials are locked.

is_whitelisted(*request*, *credentials: dict = None*) → bool
Checks if the request or given credentials are whitelisted for access.

reset_attempts(*, *ip_address: str = None*, *username: str = None*, *ip_or_username: bool = False*) → int
Resets access attempts that match the given IP address or username.

This method makes more sense for the DB backend, but as it is used by the ProxyHandler (via inherent), it needs to be defined here so we get compliant with all proxy methods.

Please overwrite it on each specialized handler as needed.

reset_logs(*, *age_days: int = None*) → int
Resets access logs that are older than given number of days.

This method makes more sense for the DB backend, but as it is used by the ProxyHandler (via inherent), it needs to be defined here so we get compliant with all proxy methods.

Please overwrite it on each specialized handler as needed.

class axes.handlers.base.**AxesHandler**
Signal bare handler implementation without any storage backend.

get_failures(*request*, *credentials: dict = None*) → int
Checks the number of failures associated to the given request and credentials.

This is a virtual method that needs an implementation in the handler subclass if the `settings.AXES_LOCK_OUT_AT_FAILURE` flag is set to `True`.

user_logged_in(*sender*, *request*, *user*, ***kwargs*)
Handles the Django `django.contrib.auth.signals.user_logged_in` authentication signal.

user_logged_out(*sender*, *request*, *user*, ***kwargs*)
Handles the Django `django.contrib.auth.signals.user_logged_out` authentication signal.

user_login_failed(*sender*, *credentials: dict*, *request=None*, ***kwargs*)
Handles the Django `django.contrib.auth.signals.user_login_failed` authentication signal.

class axes.backends.**AxesBackend**
Bases: `django.contrib.auth.backends.ModelBackend`

Authentication backend class that forbids login attempts for locked out users.

Use this class as the first item of `AUTHENTICATION_BACKENDS` to prevent locked out users from being logged in by the Django authentication flow.

Note: This backend does not log your user in. It monitors login attempts. Authentication is handled by the following backends that are configured in `AUTHENTICATION_BACKENDS`.

---

class axes.middleware.**AxesMiddleware**(*get_response: Callable*)
Middleware that calculates necessary HTTP request attributes for attempt monitoring and maps lockout signals into readable HTTP 403 Forbidden responses.

If a project uses `django rest framework` then the middleware updates the request and checks whether the limit has been exceeded. It's needed only for integration with DRF because it uses its own request object.

This middleware recognizes a logout monitoring flag in the request and and uses the `axes.helpers.get_lockout_response` handler for returning customizable and context aware lockout message to the end user if necessary.

To customize the lockout handling behaviour further, you can subclass this middleware and change the `__call__` method to your own liking.

Please see the following configuration flags before customizing this handler:

- `AXES_LOCKOUT_TEMPLATE`,
- `AXES_LOCKOUT_URL`,
- `AXES_COOLOFF_MESSAGE`, and
- `AXES_PERMALOCK_MESSAGE`.

1.9 Development

You can contribute to this project forking it from GitHub and sending pull requests.

First fork the repository and then clone it:

```
$ git clone git@github.com:<you>/django-axes.git
```

Initialize a virtual environment for development purposes:

```
$ mkdir -p ~/.virtualenvs
$ python3 -m venv ~/.virtualenvs/django-axes
$ source ~/.virtualenvs/django-axes/bin/activate
```

Then install the necessary requirements:

```
$ cd django-axes
$ pip install -r requirements.txt
```

Unit tests are located in the `axes/tests` folder and can be easily run with the pytest tool:

```
$ pytest
```

Prospector runs a number of source code style, safety, and complexity checks:

```
$ prospector
```

Mypy runs static typing checks to verify the source code type annotations and correctness:

```
$ mypy .
```

Before committing, you can run all the above tests against all supported Python and Django versions with tox:

```
$ tox
```

Tox runs the same test set that is run by GitHub Actions, and your code should be good to go if it passes.

If you wish to limit the testing to specific environment(s), you can parametrize the tox run:

```
$ tox -e py39-django22
```

After you have pushed your changes, open a pull request on GitHub for getting your code upstreamed.

1.10 Changes

1.10.1 5.26.0 (2021-10-11)

- Fix `AXES_USERNAME_CALLABLE` not receiving `credentials` attribute in Axes middleware lockout response when user is locked out. [rootart]

1.10.2 5.25.0 (2021-09-19)

- Fix duplicated AccessAttempts with updated database model `unique_together` constraints and data and schema migration. [PetrDlouhy]

1.10.3 5.24.0 (2021-09-09)

- Use atomic transaction for updating AccessAttempts in database handler. [okapies]

1.10.4 5.23.0 (2021-09-02)

- Pass `request` as argument to `AXES_CLIENT_STR_CALLABLE`. [sarahboyce]

1.10.5 5.22.0 (2021-08-31)

- Improve `failures_since_start` handling by moving the counter incrementation from non-atomic Python code call to atomic database function. [okapies]
- Add publicly available `request.axes_failures_since_start` attribute. [okapies]

1.10.6 5.21.0 (2021-08-19)

- Add configurable lockout HTTP status code responses with the new `AXES_HTTP_RESPONSE_CODE` setting. [phil-bell]

1.10.7 5.20.0 (2021-06-29)

- Improve race condition handling in e.g. multi-process environments by using `get_or_create` for access attempt fetching and updates. [uli-klank]

1.10.8 5.19.0 (2021-06-16)

- Add Polish locale. [Quadric]

1.10.9 5.18.0 (2021-06-09)

- Fix `default_auto_field` warning. [zkanda]

1.10.10 5.17.0 (2021-06-05)

- Fix `default_app_config` deprecation. Django 3.2 automatically detects `AppConfig` and therefore this setting is no longer required. [nikolaik]

1.10.11 5.16.0 (2021-05-19)

- Add `AXES_CLIENT_STR_CALLABLE` setting. [smtydn]

1.10.12 5.15.0 (2021-05-03)

- Add option to cleanse sensitive GET and POST params in database handler with the `AXES_SENSITIVE_PARAMETERS` setting. [mcoconnor]

1.10.13 5.14.0 (2021-04-06)

- Improve message formatting for lockout message and translations. [ashokdelphia]
- Remove support for Django 3.0. [hramezani]
- Add support for Django 3.2. [hramezani]

1.10.14 5.13.1 (2021-02-22)

- Default `AXES_VERBOSE` to `AXES_ENABLED` configuration setting, disabling verbose startup logging when Axes itself is disabled. [christianbundy]
- Update documentation. [KStenK]

1.10.15 5.13.0 (2021-02-15)

- Add support for resetting attempts with cache backend. [nattyg93]

1.10.16 5.12.0 (2021-01-07)

- Clean up test structure and migrate tests outside the main package for a smaller wheel distributions. [aleksihakli]
- Move configuration to pyproject.toml for cleaner layout. [aleksihakli]

- Clean up test settings override configuration. [hramezani]

1.10.17 5.11.1 (2021-01-06)

- Fix cache entry creations for None username. [cabarnes]

1.10.18 5.11.0 (2021-01-05)

- Add lockout view CORS support with `AXES_ALLOWED_CORS_ORIGINS` configuration flag. [vladox]
- Add missing `@wraps` decorator to `axes.decorators.axes_dispatch`. [aleksihakli]

1.10.19 5.10.1 (2021-01-04)

- Add `DEFAULT_AUTO_FIELD` to test settings. [hramezani]
- Fix documentation language. [danielquinn]
- Fix Python package version specifiers and remove redundant imports. [aleksihakli]

1.10.20 5.10.0 (2020-12-18)

- Deprecate stock DRF support from 5.8.0, require users to set it up per project. Check the documentation for more information. [aleksihakli]

1.10.21 5.9.1 (2020-12-02)

- Move tests to GitHub Actions [jezdez]
- Fix running Axes code in middleware when `AXES_ENABLED` is `False`. [ashokdelphia]

1.10.22 5.9.0 (2020-11-05)

- Add Python 3.9 support. [hramezani]
- Prevent `AccessAttempt` creation with database handler when username is not set and `AXES_ONLY_USER_FAILURES` setting is not set. [hramezani]

1.10.23 5.8.0 (2020-10-16)

- Improve Django REST Framework (DRF) integration. [Anatoly]

1.10.24 5.7.1 (2020-09-27)

- Adjust settings import and handling chain for cleaner module import and invocation order. [aleksihakli]
- Adjust the use of `AXES_ENABLED` flag so that imports are always done the same way and initial log is written regardless of the setting and it only affects code that is decorated or wrapped with `toggleable`. [alekshakli]

1.10.25 5.7.0 (2020-09-26)

- Deprecate `AXES_LOGGER` Axes setting and move to `__name__` based logging and fully qualified Python module name log identifiers. [aleksihakli]

1.10.26 5.6.2 (2020-09-20)

- Fix regression in `axes_reset_user` management command. [aleksihakli]

1.10.27 5.6.1 (2020-09-17)

- Improve test dependency management and upgrade black code formatter. [smithdc1]

1.10.28 5.6.0 (2020-09-12)

- Add proper development `subTest` support via `pytest-subtests` package. [smithdc1]
- Deprecate `django-appconf` and use plain settings for Axes. [aleksihakli]

1.10.29 5.5.2 (2020-09-11)

- Update deprecating use of the `request.is_ajax` method. [smithdc1]

1.10.30 5.5.1 (2020-09-10)

- Update deprecated uses of Django modules and members. [smithdc1]

1.10.31 5.5.0 (2020-08-21)

- Add support for locking requests based on username OR IP address with inclusive or using the `LOCK_OUT_BY_USER_OR_IP` flag. [PetrDlouhy]
- Deprecate Signal `providing_args` for Django 3.1 support. [coredumperror]

1.10.32 5.4.3 (2020-08-06)

- Add Django 3.1 support. [hramezani]

1.10.33 5.4.2 (2020-07-28)

- Add ABC or abstract base class implementation for handlers. [jorlugaqui]

1.10.34 5.4.1 (2020-07-03)

- Fix code styling for linters. [aleksihakli]

1.10.35 5.4.0 (2020-07-03)

- Propagate username to lockout view in URL parameters. [PetrDlouhy]
- Update CAPTCHA examples. [PetrDlouhy]
- Upgrade django-ipware to version 3. [hramezani,mnislam01]

1.10.36 5.3.5 (2020-07-02)

- Restrict ipware version for version compatibility. [aleksihakli]

1.10.37 5.3.4 (2020-06-09)

- Deprecate Django 1.11 LTS support. [aleksihakli]

1.10.38 5.3.3 (2020-05-22)

- Fix `AXES_ONLY_ADMIN_SITE` functionality when no default admin site is defined in the URL configuration. [igor-shevchenko]

1.10.39 5.3.2 (2020-05-15)

- Fix AppConf settings prefix for Fargate. [marksweb]

1.10.40 5.3.1 (2020-03-23)

- Fix null byte ValueError bug in ORM. [ddimmich]

1.10.41 5.3.0 (2020-03-10)

- Improve Django REST Framework compatibility. [I0x4dI]

1.10.42 5.2.2 (2020-01-08)

- Add missing proxy implementation for `axes.handlers.proxy.AxesProxyHandler.get_failures`. [aleksihakli]

1.10.43 5.2.1 (2020-01-08)

- Add django-reversion compatibility notes. [mark-mishyn]
- Add pluggable lockout responses and the `AXES_LOCKOUT_CALLABLE` configuration flag. [aleksihakli]

1.10.44 5.2.0 (2020-01-01)

- Add a test handler. [aidanlister]

1.10.45 5.1.0 (2019-12-29)

- Add pluggable user account whitelisting and the `AXES_WHITELIST_CALLABLE` configuration flag. [aleksihakli]

1.10.46 5.0.20 (2019-12-01)

- Fix django-allauth compatibility issue. [hramezani]
- Improve tests for login attempt monitoring. [hramezani]
- Add reverse proxy documentation. [ckcollab]
- Update OAuth documentation examples. [aleksihakli]

1.10.47 5.0.19 (2019-11-06)

- Optimize access attempt fetching in database handler. [hramezani]
- Optimize request data fetching in proxy handler. [hramezani]

1.10.48 5.0.18 (2019-10-17)

- Add `cooloff_timedelta` context variable to lockout responses. [jstockwin]

1.10.49 5.0.17 (2019-10-15)

- Safer string formatting for user input. [aleksihakli]

1.10.50 5.0.16 (2019-10-15)

- Fix string formatting bug in logging. [zerolab]

1.10.51 5.0.15 (2019-10-09)

- Add `AXES_ENABLE_ADMIN` flag. [flannelhead]

1.10.52 5.0.14 (2019-09-28)

- Docs, CI pipeline, and code formatting improvements [aleksihakli]

1.10.53 5.0.13 (2019-08-30)

- Python 3.8 and PyPy support. [aleksihakli]
- Migrate to `setuptools_scm` and automatic versioning. [aleksihakli]

1.10.54 5.0.12 (2019-08-05)

- Support callables for `AXES_COOLOFF_TIME` setting. [DariaPlotnikova]

1.10.55 5.0.11 (2019-07-25)

- Fix typo in rST formatting that prevented 5.0.10 release to PyPI. [aleksihakli]

1.10.56 5.0.10 (2019-07-25)

- Refactor type checks for `axes.helpers.get_client_cache_key` for framework compatibility, fixes #471. [aleksihakli]

1.10.57 5.0.9 (2019-07-11)

- Add better handling for attempt and log resets by moving them into handlers which allows customization and more configurability. Unimplemented handlers raise `NotImplementedError` by default. [aleksihakli]
- Add Python 3.8 dev version and PyPy to the Travis test matrix. [aleksihakli]

1.10.58 5.0.8 (2019-07-09)

- Add `AXES_ONLY_ADMIN_SITE` flag for only running Axes on admin site. [hramezani]
- Add `axes_reset_logs` command for removing old AccessLog records. [tlebrize]
- Allow `AxesBackend` subclasses to pass the `axes.W003` system check. [adamchainz]

1.10.59 5.0.7 (2019-06-14)

- Fix lockout message showing when lockout is disabled with the `AXES_LOCK_OUT_AT_FAILURE` setting. [mogzol]
- Add support for callable `AXES_FAILURE_LIMIT` setting. [bbayles]

1.10.60 5.0.6 (2019-05-25)

- Deprecate `AXES_DISABLE_SUCCESS_ACCESS_LOG` flag in favour of `AXES_DISABLE_ACCESS_LOG` which has mostly the same functionality. Update documentation to better reflect the behaviour of the flag. [aleksihakli]

1.10.61 5.0.5 (2019-05-19)

- Change the lockout response calculation to request flagging instead of exception throwing in the signal handler and middleware. Move request attribute calculation from middleware to handler layer. Deprecate `axes.request.AxesHttpRequest` object type definition. [aleksihakli]
- Deprecate the old version 4.x `axes.backends.AxesModelBackend` class. [aleksihakli]
- Improve documentation on attempt tracking, resets, Axes customization, project and component compatibility and integrations, and other things. [aleksihakli]

1.10.62 5.0.4 (2019-05-09)

- Fix regression with OAuth2 authentication backends not having remote IP addresses set and throwing an exception in cache key calculation. [aleksihakli]

1.10.63 5.0.3 (2019-05-08)

- Fix `django.contrib.auth` module `login` and `logout` functionality so that they work with the handlers without the an `AxesHttpRequest` to improve cross compatibility with other Django applications. [aleksihakli]
- Change IP address resolution to allow empty or missing addresses. [aleksihakli]

- Add error logging for missing request attributes in the handler layer so that users get better indicators of misconfigured applications. [aleksihakli]

1.10.64 5.0.2 (2019-05-07)

- Add `AXES_ENABLED` setting for disabling Axes with e.g. tests that use Django test client `login`, `logout`, and `force_login` methods, which do not supply the `request` argument to views, preventing Axes from functioning correctly in certain test setups. [aleksihakli]

1.10.65 5.0.1 (2019-05-03)

- Add changelog to documentation. [aleksihakli]

1.10.66 5.0 (2019-05-01)

- Deprecate Python 2.7, 3.4 and 3.5 support. [aleksihakli]
- Remove automatic decoration and monkey-patching of Django views and forms. Decorators are available for login function and method decoration as before. [aleksihakli]
- Use backend, middleware, and signal handlers for tracking login attempts and implementing user lockouts. [aleksihakli, jorlugaqui, joshua-s]
- Add `AxesDatabaseHandler`, `AxesCacheHandler`, and `AxesDummyHandler` handler backends for processing user login and logout events and failures. Handlers are configurable with the `AXES_HANDLER` setting. [aleksihakli, jorlugaqui, joshua-s]
- Improve management commands and separate commands for resetting all access attempts, attempts by IP, and attempts by username. New command names are `axes_reset`, `axes_reset_ip` and `axes_reset_username`. [aleksihakli]
- Add support for string import for `AXES_USERNAME_CALLABLE` that supports dotted paths in addition to the old callable type such as a function or a class method. [aleksihakli]
- Deprecate one argument call signature for `AXES_USERNAME_CALLABLE`. From now on, the callable needs to accept two arguments, the HttpRequest and credentials that are supplied to the Django `authenticate` method in authentication backends. [aleksihakli]
- Move `axes.attempts.is_already_locked` function to `axes.handlers.AxesProxyHandler.is_locked`. Various other previously undocumented methods have been deprecated and moved inside the project. The new documented public APIs can be considered as stable and can be safely utilized by other projects. [aleksihakli]
- Improve documentation layouting and contents. Add public API reference section. [aleksihakli]

1.10.67 4.5.4 (2019-01-15)

- Improve README and documentation [aleksihakli]

1.10.68 4.5.3 (2019-01-14)

- Remove the unused `AccessAttempt.trusted` flag from models [aleksihakli]
- Improve README and Travis CI setups [aleksihakli]

1.10.69 4.5.2 (2019-01-12)

- Added Turkish translations [obayhan]

1.10.70 4.5.1 (2019-01-11)

- Removed duplicated check that was causing issues when using APIs. [camilonova]
- Added Russian translations [lubicz-sielski]

1.10.71 4.5.0 (2018-12-25)

- Improve support for custom authentication credentials using the `AXES_USERNAME_FORM_FIELD` and `AXES_USERNAME_CALLABLE` settings. [mastacheata]
- Updated behaviour for fetching username from request or credentials: If no `AXES_USERNAME_CALLABLE` is configured, the optional `credentials` that are supplied to the axes utility methods are now the default source for client username and the HTTP request POST is the fallback for fetching the user information. `AXES_USERNAME_CALLABLE` implements an alternative signature with two arguments `request, credentials` in addition to the old `request` call argument signature in a backwards compatible fashion. [aleksihakli]
- Add official support for the Django 2.1 version and Python 3.7. [aleksihakli]
- Improve the requirements, documentation, tests, and CI setup. [aleksihakli]

1.10.72 4.4.3 (2018-12-08)

- Fix MANIFEST.in missing German translations [aleksihakli]
- Add *AXES_RESET_ON_SUCCESS* configuration flag [arjenzijlstra]

1.10.73 4.4.2 (2018-10-30)

- fix missing migration and add check to prevent it happening again. [markddavidoff]

1.10.74 4.4.1 (2018-10-24)

- Add a German translation [adonig]
- Documentation wording changes [markddavidoff]
- Use *get_client_username* in *log_user_login_failed* instead of credentials [markddavidoff]
- pin prospector to 0.12.11, and pin astroid to 1.6.5 [hsiaoyi0504]

1.10.75 4.4.0 (2018-05-26)

- Added AXES_USERNAME_CALLABLE [jaadus]

1.10.76 4.3.1 (2018-04-21)

- Change custom authentication backend failures from error to warning log level [aleksihakli]
- Set up strict code linting for CI pipeline that fails builds if linting does not pass [aleksihakli]
- Clean up old code base and tests based on linter errors [aleksihakli]

1.10.77 4.3.0 (2018-04-21)

- Refactor and clean up code layout [aleksihakli]
- Add prospector linting and code checks to toolchain [aleksihakli]
- Clean up log message formatting and refactor type checks [EvaSDK]
- Fix faulty user locking with user agent when AXES_ONLY_USER_FAILURES is set [EvaSDK]

1.10.78 4.2.1 (2018-04-18)

- Fix unicode string interpolation on Python 2.7 [aleksihakli]

1.10.79 4.2.0 (2018-04-13)

- Add configuration flags for client IP resolving [aleksihakli]
- Add AxesModelBackend authentication backend [markdaviddoff]

1.10.80 4.1.0 (2018-02-18)

- Add AXES_CACHE setting for configuring *axes* specific caching. [JWvDronkelaar]
- Add checks and tests for faulty LocMemCache usage in application setup. [aleksihakli]

1.10.81 4.0.2 (2018-01-19)

- Improve Windows compatibility on Python < 3.4 by utilizing win_inet_pton [hsiaoyi0504]
- Add documentation on django-allauth integration [grucha]
- Add documentation on known AccessAttempt caching configuration problems when using axes with the *django.core.cache.backends.locmem.LocMemCache* [aleksihakli]
- Refactor and improve existing AccessAttempt cache reset utility [aleksihakli]

1.10.82 4.0.1 (2017-12-19)

- Fixes issue when not using *AXES_USERNAME_FORM_FIELD* [camilonova]

1.10.83 4.0.0 (2017-12-18)

- *BREAKING CHANGES. AXES_BEHIND_REVERSE_PROXY AXES_REVERSE_PROXY_HEADER AXES_NUM_PROXIES* were removed in order to use *django-ipware* to get the user ip address [camilonova]
- Added support for custom username field [kakulukia]
- Customizing Axes doc updated [pckapps]
- Remove filtering by username [camilonova]
- Fixed logging failed attempts to authenticate using a custom authentication backend. [D3X]

1.10.84 3.0.3 (2017-11-23)

- Test against Python 2.7. [mbaechtold]
- Test against Python 3.4. [pope1ni]

1.10.85 3.0.2 (2017-11-21)

- Added form_invalid decorator. Fixes #265 [camilonova]

1.10.86 3.0.1 (2017-11-17)

- Fix DeprecationWarning for logger warning [richardowen]
- Fixes global lockout possibility [joeribekker]
- Changed the way output is handled in the management commands [ataylor32]

1.10.87 3.0.0 (2017-11-17)

- BREAKING CHANGES. Support for Django >= 1.11 and signals, see issue #215. Drop support for Python < 3.6 [camilonova]

1.10.88 2.3.3 (2017-07-20)

- Many tweaks and handles successful AJAX logins. [Jack Sullivan]
- Add tests for proxy number parametrization [aleksihakli]
- Add AXES_NUM_PROXIES setting [aleksihakli]
- Log failed access attempts regardless of settings [jimr]
- Updated configuration docs to include AXES_IP_WHITELIST [Minkey27]
- Add test for get_cache_key function [jorlugaqui]
- Delete cache key in reset command line [jorlugaqui]
- Add signals for setting/deleting cache keys [jorlugaqui]

1.10.89 2.3.2 (2016-11-24)

- Only look for lockable users on a POST [schinckel]
- Fix and add tests for IPv4 and IPv6 parsing [aleksihakli]

1.10.90 2.3.1 (2016-11-12)

- Added settings for disabling success accesslogs [Minkey27]
- Fixed illegal IP address string passed to inet_pton [samkuehn]

1.10.91 2.3.0 (2016-11-04)

- Fixed `axes_reset` management command to skip "ip" prefix to command arguments. [EvaMarques]
- Added `axes_reset_user` management command to reset lockouts and failed login records for given users. [vladimirnani]
- Fixed Travis-PyPI release configuration. [jezdez]
- Make IP position argument optional. [aredalen]
- Added possibility to disable access log [svenhertle]
- Fix for IIS used as reverse proxy adding port number [Dmitri-Sintsov]
- Made the signal race condition safe. [Minkey27]
- Added AXES_ONLY_USER_FAILURES to support only looking at the user ID. [lip77us]

1.10.92 2.2.0 (2016-07-20)

- Improve the logic when using a reverse proxy to avoid possible attacks. [camilonova]

1.10.93 2.1.0 (2016-07-14)

- Add *default_app_config* so you can just use *axes* in *INSTALLED_APPS* [vdboor]

1.10.94 2.0.0 (2016-06-24)

- Removed middleware to use app_config [camilonova]
- Lots of cleaning [camilonova]
- Improved test suite and versions [camilonova]

1.10.95 1.7.0 (2016-06-10)

- Use render shortcut for rendering LOCKOUT_TEMPLATE [Radoslaw Luter]
- Added app_label for RemovedInDjango19Warning [yograterol]
- Add iso8601 translator. [mullakhmetov]
- Edit json response. Context now contains ISO 8601 formatted cooloff time [mullakhmetov]
- Add json response and iso8601 tests. [mullakhmetov]
- Fixes issue 162: UnicodeDecodeError on pip install [joeribekker]

- Added AXES_NEVER_LOCKOUT_WHITELIST option to prevent certain IPs from being locked out. [joeribekker]

1.10.96 1.6.1 (2016-05-13)

- Fixes whitelist check when BEHIND_REVERSE_PROXY [Patrick Hagemeister]
- Made migrations py3 compatible [mvdwaeter]
- Fixing #126, possibly breaking compatibility with Django<=1.7 [int-ua]
- Add note for upgrading users about new migration files [kelseyq]
- Fixes #148 [camilonova]
- Decorate auth_views.login only once [teeberg]
- Set IP public/private classifier to be compliant with RFC 1918. [SilasX]
- Issue #155. Lockout response status code changed to 403. [Arthur Mullahmetov]
- BUGFIX: Missing migration [smeinel]

1.10.97 1.6.0 (2016-01-07)

- Stopped using render_to_response so that other template engines work [tarkatronic]
- Improved performance & DoS prevention on query2str [tarkatronic]
- Immediately return from is_already_locked if the user is not lockable [jdunck]
- Iterate over ip addresses only once [annp89]
- added initial migration files to support django 1.7 &up. Upgrading users should run migrate –fake-initial after update [ibaguio]
- Add db indexes to CommonAccess model [Schweigi]

1.10.98 1.5.0 (2015-09-11)

- Fix #_get_user_attempts to include username when filtering AccessAttempts if AXES_LOCK_OUT_BY_COMBINATION_USER_AND_IP is True [afioca]

1.10.99 1.4.0 (2015-08-09)

- Send the user_locked_out signal. Fixes #94. [toabi]

1.10.100 1.3.9 (2015-02-11)

- Python 3 fix (#104)

1.10.101 1.3.8 (2014-10-07)

- Rename GitHub organization from django-security to django-pci to emphasize focus on providing assistance with building PCI compliant websites with Django. [aclark4life]

1.10.102 1.3.7 (2014-10-05)

- Explain common issues where Axes fails silently [cericoda]
- Allow for user-defined username field for lookup in POST data [SteveByerly]
- Log out only if user was logged in [zoten]
- Support for floats in cooloff time (i.e: 0.1 == 6 minutes) [marianov]
- Limit amount of POST data logged (#73). Limiting the length of value is not enough, as there could be arbitrary number of them, or very long key names. [peterkuma]
- Improve get_ip to try for real ip address [7wonders]
- Change IPAddressField to GenericIPAddressField. When using a PostgreSQL database and the client does not pass an IP address you get an inet error. This is a known problem with PostgreSQL and the IPAddressField. https://code.djangoproject.com/ticket/5622. It can be fixed by using a GenericIPAddressField instead. [polvoblanco]
- Get first X-Forwarded-For IP [tutumcloud]
- White listing IP addresses behind reverse proxy. Allowing some IP addresses to have direct access to the app even if they are behind a reverse proxy. Those IP addresses must still be on a white list. [ericbulloch]
- Reduce logging of reverse proxy IP lookup and use configured logger. Fixes #76. Instead of logging the notice that django.axes looks for a HTTP header set by a reverse proxy on each attempt, just log it one-time on first module import. Also use the configured logger (by default axes.watch_login) for the message to be more consistent in logging. [eht16]
- Limit the length of the values logged into the database. Refs #73 [camilonova]
- Refactored tests to be more stable and faster [camilonova]
- Clean client references [camilonova]
- Fixed admin login url [camilonova]
- Added django 1.7 for testing [camilonova]
- Travis file cleanup [camilonova]
- Remove hardcoded url path [camilonova]
- Fixing tests for django 1.7 [Andrew-Crosio]
- Fix for django 1.7 exception not existing [Andrew-Crosio]
- Removed python 2.6 from testing [camilonova]
- Use django built-in six version [camilonova]

- Added six as requirement [camilonova]
- Added python 2.6 for travis testing [camilonova]
- Replaced u string literal prefixes with six.u() calls [amrhassan]
- Fixes object type issue, response is not an string [camilonova]
- Python 3 compatibility fix for db_reset [nicois]
- Added example project and helper scripts [barseghyanartur]
- Admin command to list login attemps [marianov]
- Replaced six imports with django.utils.six ones [amrhassan]
- Replaced u string literal prefixes with six.u() calls to make it compatible with Python 3.2 [amrhassan]
- Replaced *assertIn's and 'assertNotIn's with 'assertContains* and *assertNotContains* [fcurella]
- Added py3k to travis [fcurella]
- Update test cases to be python3 compatible [nicois]
- Python 3 compatibility fix for db_reset [nicois]
- Removed trash from example urls [barseghyanartur]
- Added django installer [barseghyanartur]
- Added example project and helper scripts [barseghyanartur]

1.10.103 1.3.6 (2013-11-23)

- Added AttributeError in case get_profile doesn't exist [camilonova]
- Improved axes_reset command [camilonova]

1.10.104 1.3.5 (2013-11-01)

- Fix an issue with __version__ loading the wrong version [graingert]

1.10.105 1.3.4 (2013-11-01)

- Update README.rst for PyPI [marty, camilonova, graingert]
- Add cooloff period [visualspace]

1.10.106 1.3.3 (2013-07-05)

- Added 'username' field to the Admin table [bkvirendra]
- Removed fallback logging creation since logging cames by default on django 1.4 or later, if you don't have it is because you explicitly wanted. Fixes #45 [camilonova]

1.10.107 1.3.2 (2013-03-28)

- Fix an issue when a user logout [camilonova]
- Match pypi version [camilonova]
- Better User model import method [camilonova]
- Use only one place to get the version number [camilonova]
- Fixed an issue when a user on django 1.4 logout [camilonova]
- Handle exception if there is not user profile model set [camilonova]
- Made some cleanup and remove a pokemon exception handling [camilonova]
- Improved tests so it really looks for the rabbit in the hole [camilonova]
- Match pypi version [camilonova]

1.10.108 1.3.1 (2013-03-19)

- Add support for Django 1.5 [camilonova]

1.10.109 1.3.0 (2013-02-27)

- Bug fix: get_version() format string [csghormley]

1.10.110 1.2.9 (2013-02-20)

- Add to and improve test cases [camilonova]

1.10.111 1.2.8 (2013-01-23)

- Increased http accept header length [jslatts]

1.10.112 1.2.7 (2013-01-17)

- Reverse proxy support [rmagee]
- Clean up README [martey]

1.10.113 1.2.6 (2012-12-04)

- Remove unused import [aclark4life]

1.10.114 1.2.5 (2012-11-28)

- Fix setup.py [aclark4life]
- Added ability to flag user accounts as unlockable. [kencochrane]
- Added ipaddress as a param to the user_locked_out signal. [kencochrane]
- Added a signal receiver for user_logged_out. [kencochrane]
- Added a signal for when a user gets locked out. [kencochrane]
- Added AccessLog model to log all access attempts. [kencochrane]

CHAPTER 2

Indices and tables

- search

Python Module Index

a

Index